# MapMatching4GMNS

Please send your comments to [xzhou74@asu.edu](mailto:xzhou74@asu.edu) if you have any suggestions and questions.

## 1. Introduction

Based on input network and given GPS trajectory data, the map-matching program of MapMatching4GMNS aims to find the most likely route in terms of node sequence in the underlying network, with the following data flow chart.

GMNS: General Modeling Network Specification (GMNS) (<https://github.com/zephyr-data-specs/GMNS>)

## 2. Installation

MapMatching4GMNS has been released on [PyPI](https://test.pypi.org/project/MapMatching4GMNS/), and can be installed using

$ pip install MapMatching4GMNS

If you need a newly version of MapMatching4GMNS,

$ pip **install** MapMatching4GMNS –upgrade

**Windows Users**  
You do not need to install Microsoft Visual C++ Redistributable for Visual Studio; only copy the missing dependency libraries from the Dependent\_libraries\_missing\_in\_windows\_system folder to your computer path(C:\Windows\System32).  
Note: If your windows system has existed some dependencies in the C:\Windows\System32, you only need to copy the dependency libraries that are not.

## Getting Started

### Download the Test Data Set

A sample data set with six different networks are provided. You can manually retrieve each individual test network from [here](https://github.com/asu-trans-ai-lab/osm_test_data_set//datasets/map_matchindata).\*

**from MapMatching4GMNS import**

#First, download the **input data** of the test: node.csv, link.csv and trace.csv from Github.

**MapMatching4GMNS.download\_sample\_data\_sets\_from\_network()**

#If the online download fails**,** Please download manually the **input** **data** **from** <https://github.com/asu-trans-ai-lab/osm_test_data_set/map_matching/>.

#Second, **call** the mapmatching4gmns library **to** calculate **and** **output** the result **in** the **current** directory.

**MapMatching4GMNS.map\_match()**

## 3. Data flow

|  |  |  |  |
| --- | --- | --- | --- |
|  | **files** | **Data Source** | **Visualization** |
| GMNS network input | node.csv, link.csv | [Osm4GMNS](https://osm2gmns.readthedocs.io/en/latest/) | [QGIS](https://www.qgis.org/en/site/), [web interface for GMNS](https://asu-trans-ai-lab.github.io/index.html#/) |
| Location sequence data input | trace.csv | GPS traces downloaded from OpenStreetMap, e.g., using the script at <https://github.com/asu-trans-ai-lab/MapMatching4GMNS/blob/master/release/get_gps_trace.py> | QGIS |
| Map-matched output | route.csv |  | QGIS |

The M4G program can be executed using one of the following 2 different modes: The Python package mode is mainly used in an effective integration with other packages such as OSM2GMNS and Path2GMNS. The windows executable mode aims to help users generate the results directly without replying on the python environment.

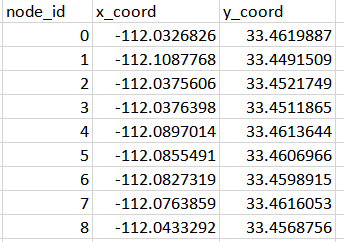
**Mode 1: Python package and test script**: <https://github.com/asu-trans-ai-lab/MapMatching4GMNS/blob/master/MapMatching4GMNS.ipynb>

**Mode 2: Windows Executable: M4G.exe** can be found from

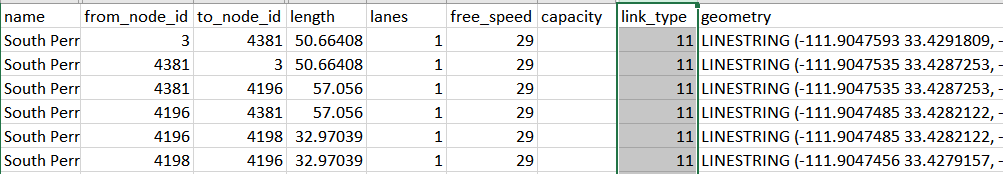
<https://github.com/asu-trans-ai-lab/MapMatching4GMNS/tree/master/release>

## 4. File description

**File node.csv** gives essential node information of the underlying network in GMNS format, including node\_id, x\_coord and y\_coord.

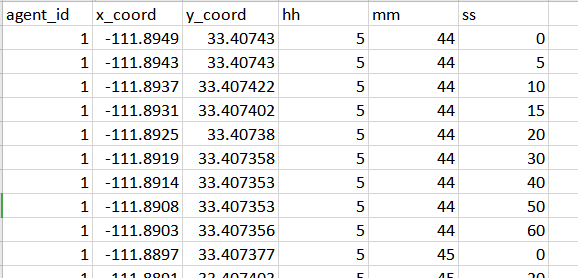


**File link.csv** should include essential link information of the underlying (subarea) network, including from\_node\_id, to\_node\_id, length and geometry.



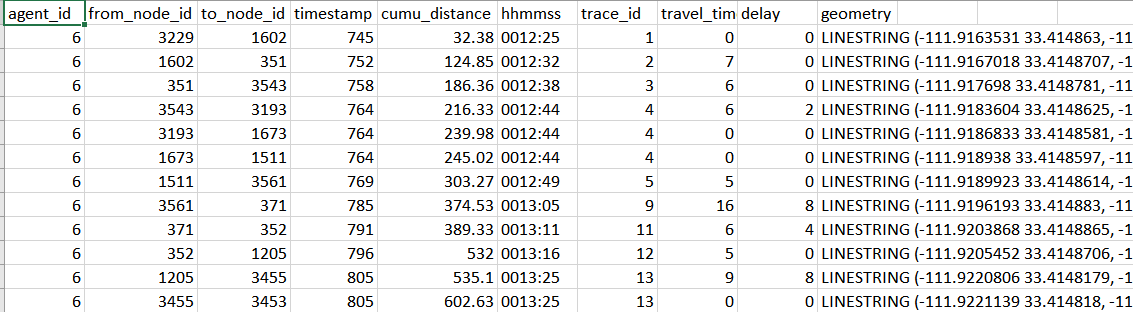
**Input trace file** as

The agent id is GPS trace id, x\_coord and y\_coord should be consistent to the network coordinate defined in node.csv and link.cvs. Fields hh mm and ss correspond the hour, minute and second for the related GPS timestamp. We use separate columns directly to avoid confusion caused by different time coding formats.



**Output file description**

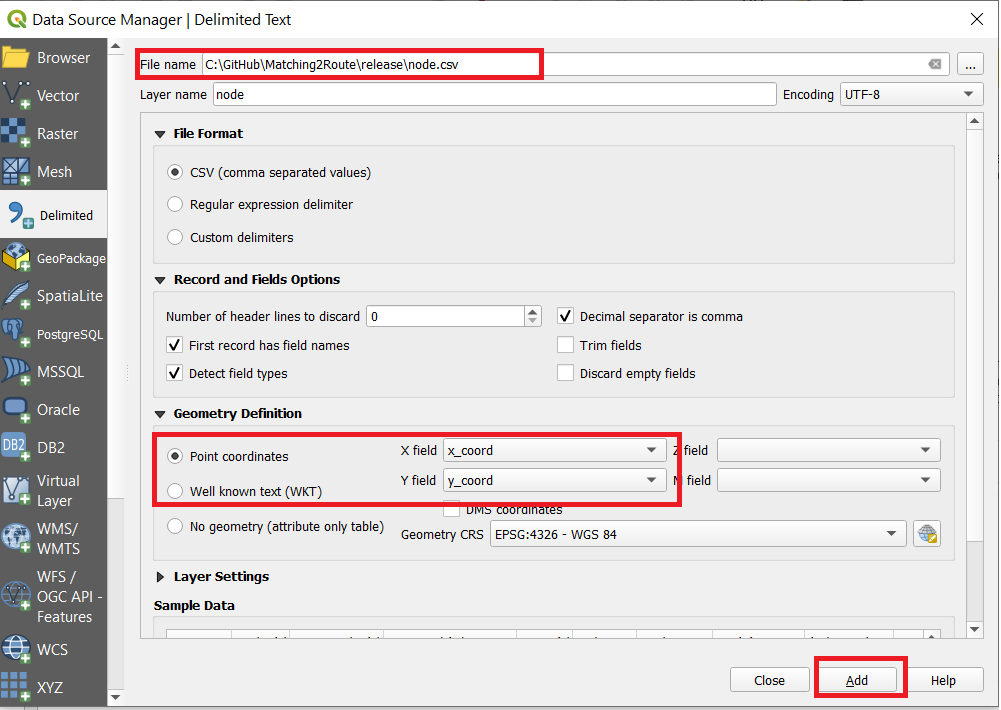
**File route.csv** describes the most-likely path for each agent based on input trajectories.

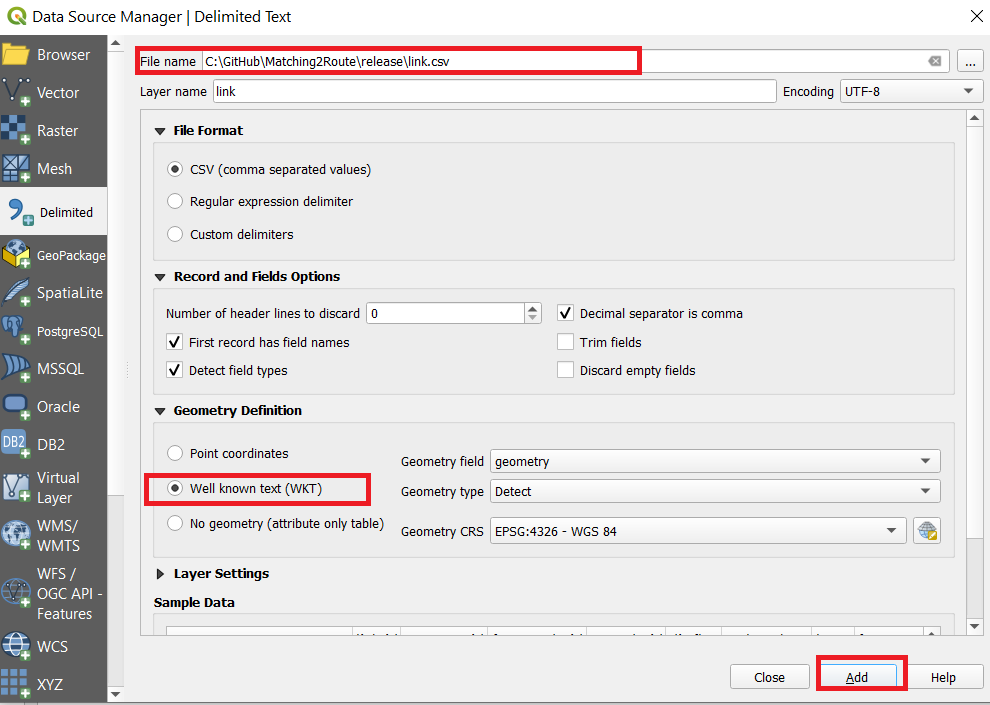


## 5. Visualization

### Step 1: Load GMNS files in QGIS

Install and open QGIS and click on menu Layer->Add->Add Delimited Text Layer. In the following dialogue box, load GMNS node.csv and link.csv, and ensure  
“point coordinates” is selected as geometry definition for node.csv wit x\_coord and y\_coord for “Geometry field”, and WKT is selected as geometry definition for link.csv.





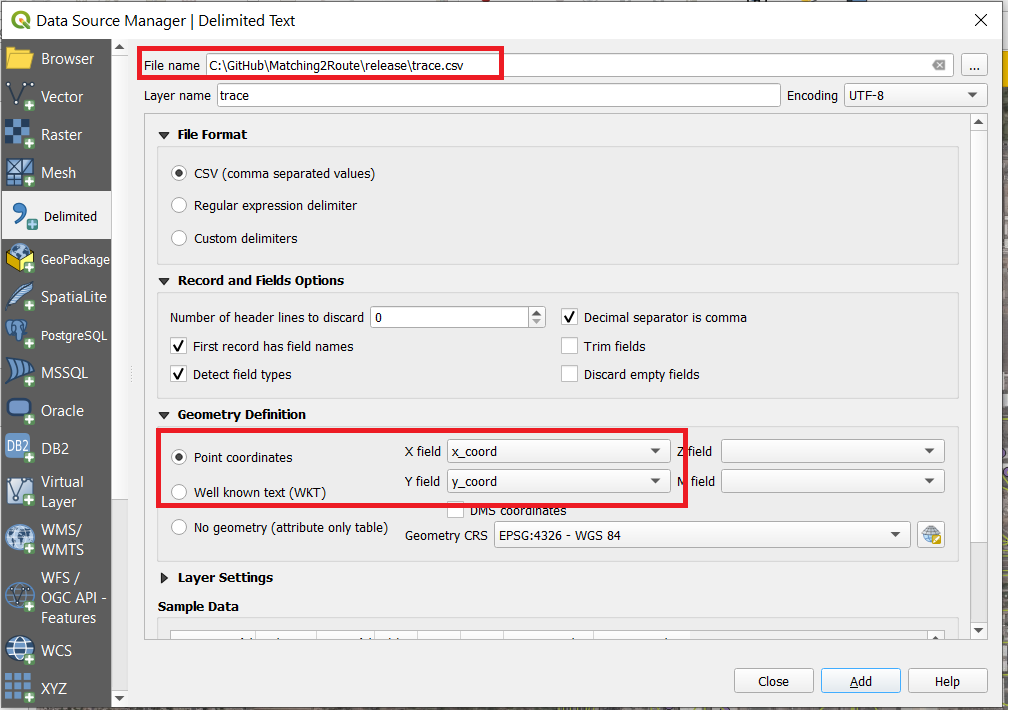
### Step 2: Load XYZ Tiles in QGIS with background maps

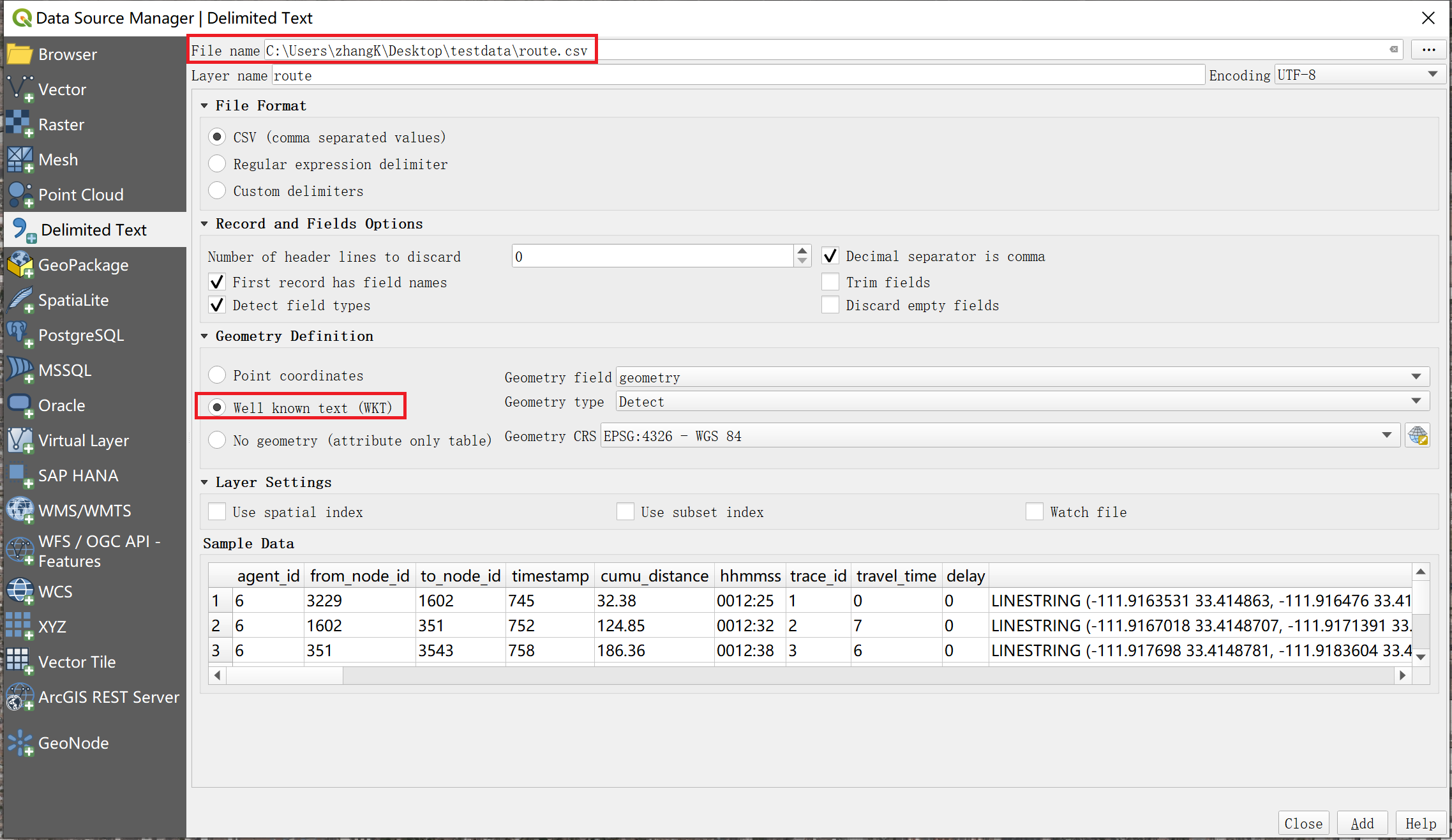
Find XYZ Tiles and double-click OpenStreetMap on Browser panel. Please move the background layer to the bottom to show the GMNS network.

Refence: <https://gis.stackexchange.com/questions/20191/adding-basemaps-from-google-or-bing-in-qgis>

### Step 3. Visualize input trace and output route files in QGIS

The ‘geometry’ field can be obtained from link.csv file. Then open this file in the same way as above. (Menu Layer->Add->Add Delimited Text Layer)







## 6. Algorithm

1. **Read standard GMNS network files** node and link files, **Read GPS trace.csv** file

* Note: the M4G program will convert trace.csv to input\_agent.csv for visualization in NeXTA.

1. **Construct 2d grid system** to speed up the indexing of GSP points to the network. For example, a 10x10 grid for a network of 100 K nodes could lead to 1K nodes in each cell.
2. **Identify the related subarea** for the traversed cells by each GPS trace, so only a small subset of the network will be loaded in the resulting shortest path algorithm.
3. **Identify the origin and destination** nodes in the grid for each GPS trace, in case, the GPS trace does not start from or end at a node inside the network (in this case, the boundary origin and destination nodes will be identified). The OD node identification is important to run the following shortest path algorithm.
4. **Estimate link cost** to calculate a generalized weight/cost for each link in the cell, that is, the distance from nearly GPS points to a link inside the cell.
5. Use **likely path finding algorithm** selects the least cost path with the smallest generalized cumulative cost from the beginning to the end of the GPS trace.
6. **Identify matched timestamps** of each node in the likely path
7. **Output route.csv** with **estimated link travel time and delay** based on free-flow travel time of each link along the GPS matched routes

## Reference

This code is implemented partially based on a published paper in Transportation Research Part C:

Tang J, Song Y, Miller HJ, Zhou X (2015) “Estimating the most likely space–time paths, dwell times and path uncertainties from vehicle trajectory data: A time geographic method,” *Transportation Research Part C*, <http://dx.doi.org/10.1016/j.trc.2015.08.014>